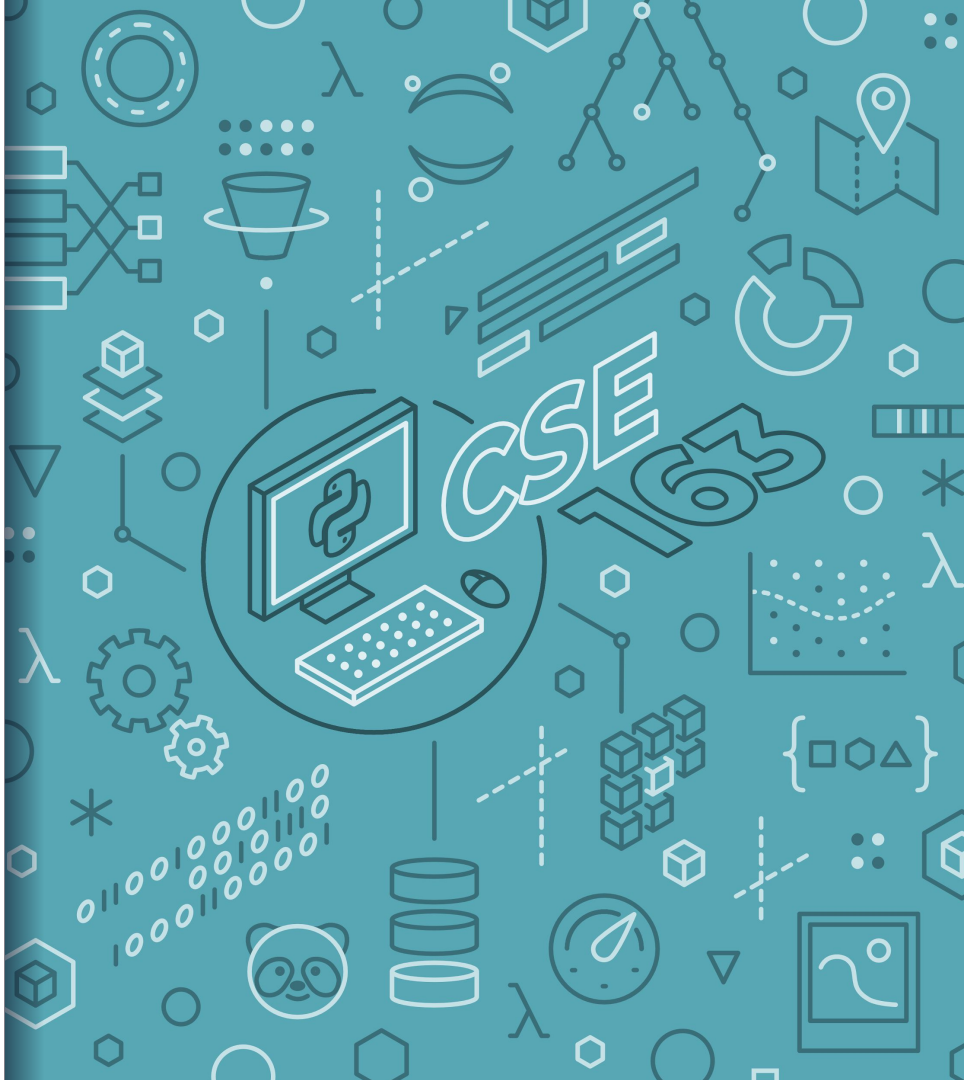
CSE 163

Classes + HW4 + Logistics

Hunter Schafer



Class

- A **class** lets you define a new object type by specifying what state and behaviors it has
- A class is a blueprint that we use to construct **instances** of the object

Here is a full class

```
class Dog:
  def __init__(self, name):
    self.name = name

  def bark(self):
    print(self.name + ': Woof')
```

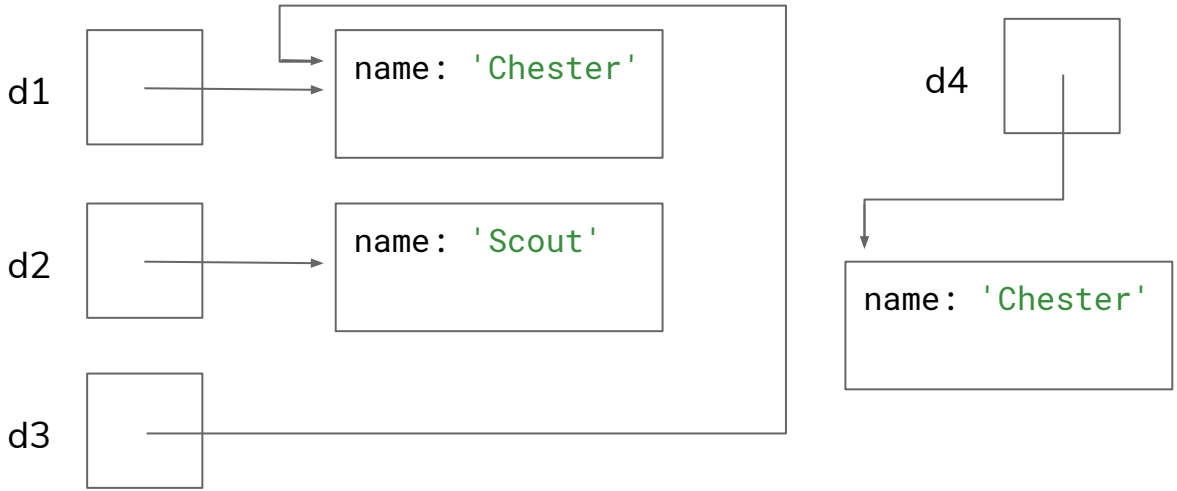
A class definition

An initializer that sets fields **(state)**

A method **(behavior)**

Building Dogs

```
d1 = Dog('Chester')
d2 = Dog('Scout')
d3 = d1
d4 = Dog('Chester')
```



Identity vs Equality

- “Is this object equal to this other object?”
  - The answer depends on what you mean by equal
- Equality can mean one of two things
  - The objects have the same identity (i.e. same person)
  - The objects have the same value (e.g. same name)

```
l1 = [1, 2, 3]
l2 = [1, 2, 3]
l3 = l1

l1 == l2  # True, same values
l1 is l2  # False, different object

l1 is l3  # True, same object
```

Identity vs Equality

- Let's try this with our Dog class

```
d1 = Dog('Chester')
d2 = Dog('Chester')
d3 = d1

d1 is d2  # False
d1 is d3  # True
d1 == d2  # False?
```

- Python doesn't know by default what equality by value means
- We have to tell it by implementing a `__eq__` method

__eq__

```
class Dog:
  def __init__(self, name):
    self.name = name

  def bark(self):
    print(self.name + ': Woof')

  def __eq__(self, other):
    return self.name == other.name



d1 = Dog('Chester')
d2 = Dog('Chester')
d1 == d2  # True, same as d1.__eq__(d2)
```

Special Methods

- The fact that you can use the same syntax for many classes is because of these “special methods”

Syntax	**Method**
`x < y`	`x.__lt__(y)`
`x == y`	`x.__eq__(y)`
`x > y`	`x.__ge__(y)`
`print(x)`	`print(x.__repr__())`
`x[i]`	`x.__getitem__(i)`
`x[i] = v`	`x.__setitem__(i, v)`

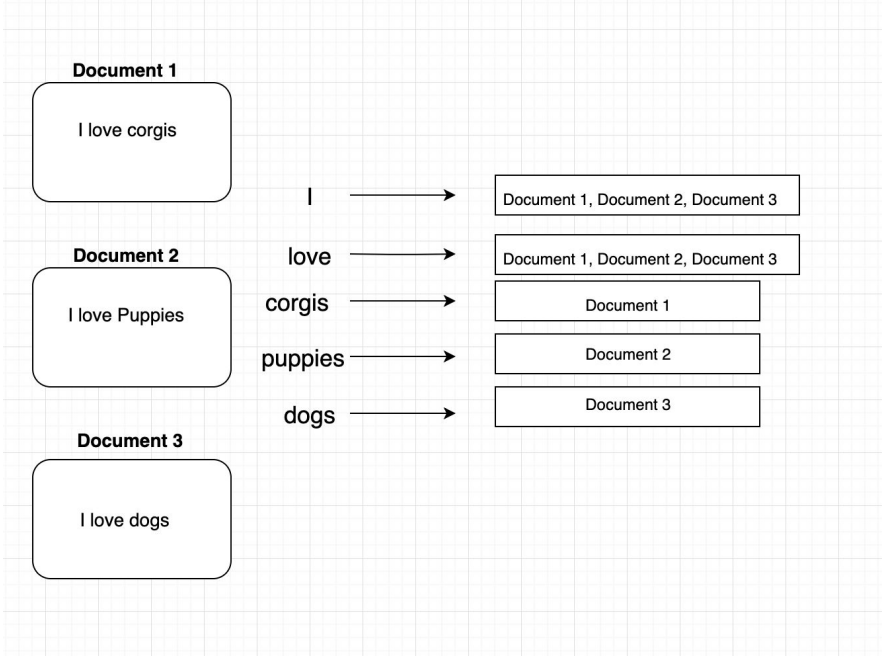
Search Engine

- HW4 involves writing a search engine from scratch
- This will mainly involve
  - Processing files
  - Writing classes
- Can be a bit complicated at first, recommend reading over these slides and the entire spec first to see how the pieces fit together, then tackle each part
- We have parts separated into a development strategy

- A search engine involves two main components
  - An **index** to efficiently find results for a query
  - A **ranking algorithm** to order the results
- We'll describe the index and searching first even though you will implementing part of the ranking algorithm first on the assignment

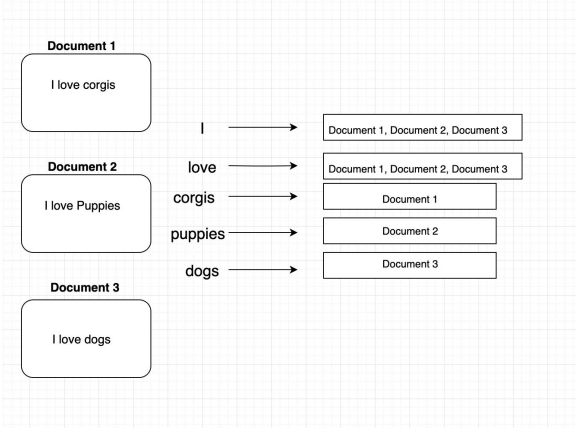
SearchEngine

- SearchEngine: A class that manages which words appear in which document
- Uses an **inverted-index**, to perform search



search

- Part 1 - Single-word query
  - search(“love”)
    - Document 1
    - Document 2
    - Document 3
- Part 2 - Multi-word query
  - search(“love corgis”)
    - Document 1
    - Document 2
    - Document 3
  - search(“corgis puppies”)
    - Document 1
    - Document 2



Document Ranking

- Once we have a set of results, need rank them by relevance
- Query: “the dogs”

dogs_rock.txt
Dogs are the most amazing pets. Dogs are way better than cats because they are the best pets.

dictionary.txt
a - An article... aardvark - An animal... ... avocado - A fruit… ... dog - The best pet… ...

- Which order should we rank the results?
  - If we did it by raw number of hits, this would almost always list the dictionary as the first result

TF-IDF

- Raw counts fails because it favors long documents that have common words
- Intuition, give higher weight to less common words and penalize common words like “the”

```
Score(“the dogs”, D) = TFIDF(“the”, D) + TFIDF(“dogs”, D)
TFIDF(t, D) = TF(t, D) * IDF(t)
TF(t, D) = (# of times t in D) / (# of words in D)
IDF(t) = (# of documents) / (# of documents that have t)
```

TF-IDF

$$TF(t, D) = \frac{\text{number of occurences of term } t \text{ in the document}}{\text{number of words in the document}}$$

$$IDF(t) = \begin{cases} 0 & t \text{ does not appear in the Search Engine} \\ ln(\frac{\text{total number of documents in Search Engine}}{\text{number of documents in Search Engine containing term } t}) & \text{otherwise} \end{cases}$$

$$TFIDF(t, D) = TF(t, D) \cdot IDF(t)$$

$$TFIDF(q, D) = \sum_{t \in q} TF(t, D) \cdot IDF(t)$$

How to compute TF-IDF

- It would be inefficient to calculate all these values from scratch every time we get a new query
- Instead, we will store all the information ahead of time
  - Takes memory and a little bit to build, but then queries are fast
- `Document` class stores Term-Frequency information
- `SearchEngine` class stores Inverse-Document-Frequency information

Testing

- Testing this assignment can be tricky with the wikipedia dataset
  - It's large, takes a few minutes to build the index
  - It's hard to know what the answer "should be"
- You are recommend to make your own directories of test text files

- In order to turn all of this in, we are asking that you turn in a zip of your hw4 folder. Instructions on website.
- It's okay to directly inspect your private fields in tests
  - We will do this for the private methods we ask you to write

Development Strategy

- Read the whole spec first
- Take notes as you're writing and try to make a mental map of how the pieces fit together
- Write down any of the special method calls we tell you about in the spec and what they do so you know when you'll use them
- Planning up front can be very helpful with this project
    - You should probably still tackle the project in the order we suggested, but thinking about the whole thing is important

Sorting

- Python has a built-in function called sorted
- Not supported by many types by default, but you can use the key parameter to help
  - Key takes a **function** that takes an object and turns it into value that is sortable.
  - Python then sorts the list based on the key value for each value in the list.

Lambdas

- It's kind of annoying to have to define a whole function just for this. It would be nice if there was a shorthand syntax

```
def get_name(d):
    return d.name

f = get_name
sorted(dogs, key=f)
```

```
f = lambda d: d.name
sorted(dogs, key=f)
```

```
sorted(dogs, key=lambda d: d.name)
```

Logistics

- Exam 1 Materials Posted
  - Next week's content, last thing on exam 1
  - Practice exam coming soon
  - Section problems, practice problems, **homework** great things to study.
- Project information released
  - Part 0 due in 3 weeks
  - Part 0 - Find dataset and come up with research questions
  - Fast turn-around for Part 1 which is outlining your methodologies, motivations, and work-plan

Calendar

21 Easter Sunday	22 Easter Monday	23	24	25	26	27
SUN 28	MON 29	TUE 30	WED May 1	THU 2 HW4 Due	FRI 3	SAT 4
5 Cinco de Mayo	6	7	8	9	10 Exam 1	11
12 Mother's Day	13	14	15	16 ~~HW5 Due~~	17 Part 0	18
19	20	21	22	23 HW6 Due	24 Part 1	25